Programmeerimise põhikursus

ITI0010

Loengu sisu

- **Üldist:**
  - Keele vahendid Javas ja C-s
  - Millised keele osad on hädavajalikud programmeerimiseks?
- **Konkreetset:**
  - Kommentaarid
  - Muutujad
  - Avaldised
  - Kontrollkonstruktsioonid
- **Kompilaator, interpretaator, käsurida, apletid, servletid:**
  - Java kompilaator ja interpretaator
  - Java programmid käsurealt, aplettide ja servlettidena

Java keele vahendid: mida leidub?

- Muutujad, avaldised:
  - Muutujad, avaldised, omistamine: täpselt nagu C
  - Tüübid: sarnased, kuid veidi teistsugused, kui C
- Kontrollkonstruktsioonid:
  - Kontrollkonstruktsioonid: if, while, for, do, ...: täpselt nagu C
  - Veakontrolli konstruktsioonid: try, catch, throw: C-s neid ei ole
  - Paralleelprotsesside konstruktsioonid: C-s neid ei ole
- Funktsioonid jms:
  - Funktsioonid: täpselt nagu C
  - Klassid: C-s neid ei ole, C++ on. Javas veidi lihtsamad kui C++
- Standardteek: väga mahukas, palju palju suurem, kui C standardteek

Java keele vahendid ja teised: vahekommentaar

- Põhikonstruktsioonid Java keeles on täpselt samad, kui C-s, C++-s, C#-s.

- C edasiarendus objekt-orienteeritud programmeerimise jaoks on C++

- Java on suuresti C++ ideede järgi ehitatud

- C# on suuresti Java ideede järgi ehitatud ja Java-ga äärmiselt sarnane.

Millistest keele vahenditest piisaks?

- Programmeerimiseks on hädavajalikud väga vähesed andmetüübid ja konstruktsioonid:
  - Täisarvud (int)
  - Massiivid
  - Omistamine
  - Aritmeetika-avaldised (+, < jms)
  - If
  - While (või hoopis for, igaühe maitse järgi)
- Kõik muud on abistavad: teevad elu lihtsamaks.

  Samas ei ole mõtet abistavate konstruktsioonidega oma elu liiga keeruliseks teha

Progr in the small, progr in the large

- **In the small:**
  - Konkreetsed algoritmid tsüklitest ja if-dest ja omistamistest jms
- **In the large:**
  - Suure programmi jaotamine funktsioonideks ja klassideks

NB! “Programming in the small” on olulisem, kui “programming in the large”.

Just to recall: program class structure

- A program is defined by a public class that takes the form:

```
public class program-name {

  optional-variable-declarations-and-subroutines

  public static void main(String[] args) {
    statements
  }

  optional-variable-declarations-and-subroutines

}
```

- NB! In principle, we could program anything by simply filling the body of "main", defining not a single extra function or class.
- However, programming in this way is extremely painful for any nontrivial programs: in practice it is crucial to split a program into many small functions and classes.

Comments: like C, but more like C++

- In C source a comment is anything between

```
/*    ....    */
```

- In Java (like C++, C# and gcc extensions to C) there is additionally a line comment: anything from

```
//
```

to the end of the line is a comment

Variables & primitive names: just like C

- `N n rate x15 a_long_name time_is_$ HelloWorld`

- `HelloWorld, helloworld, HELLOWORLD, and hElloWorLD are all distinct names`

- reserved words include: `class, public, static, if, else, while, and several dozen other words.`

- An assignment statement takes the form:

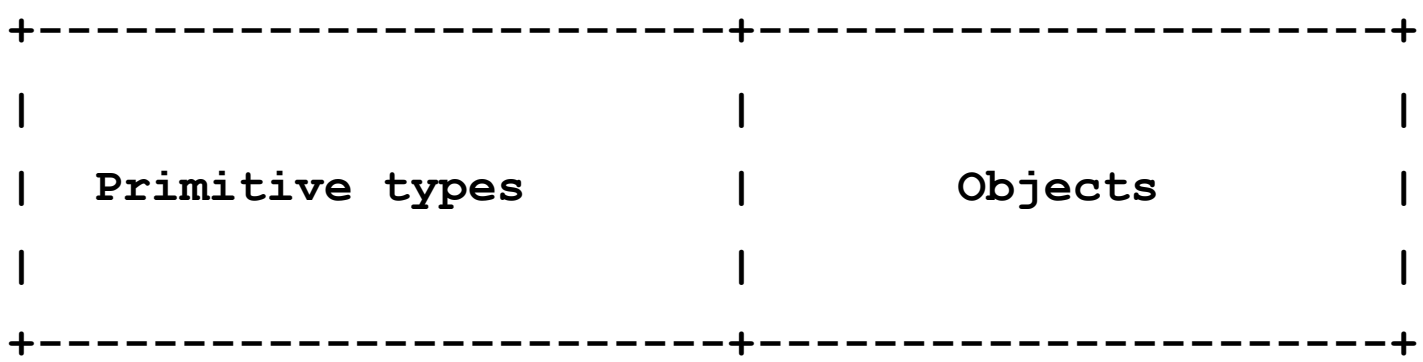
  variable = expression;

```
rate = 0.07;
interest = rate * principal;
```

Data:

- **NB! in this lecture we only use primitive types.**
- **Complex come soon.**

- Java has *very many* data types built into it, and you (as a programmer) can create as many more as you want.
- However, other than the primitive data types, *all the other data in a Java program will be represented as an object*. So there is a fundamental split in the data a Java program deals with:



All Data

Primitive TYPES

- The primitive types are named:

```
byte, short, int, long,
float, double,
char,
boolean
```

- **short** corresponds to **two bytes** (16 bits). Variables of type short have values in the range -32768 to 32767.
- **int** corresponds to **four bytes** (32 bits). Variables of type int have values in the range -2147483648 to 2147483647.
- **long** corresponds to **eight bytes** (64 bits). Variables of type long have values in the range -9223372036854775808 **to** `9223372036854775807.`
- **`boolean`** `is result of: rate > 0.05`

Variable assignment

type-name variable-name;

or

type-name variable-name = expression;

```
int N;
double x;
double rate = 0.07;
char space = ' ';
```

- You can create several variables in the same declaration, if you separate them by commas. For example:

```
double x,y;
char first = 'D',
     middle = 'J',
     last = 'E';
int i, j = 17;
```

NB! Declaring vars in Java a bit easier than in C

- In standard C it is only OK to declare variables at the beginning of a block, like this:

```
{
  int x;
  int y;
  x = 5;
  y = x + 5;
}
```

- In Java vars may be declared anywhere, like this:

```
{
  int x;
  x = 5;
  int y;
  y = x + 5;
}
```

Block

- The block is the simplest type of statement. Its purpose is simply to group a sequence of statements into a single statement. The format of a block is:

```
{
    statements
}
```

- Here are two examples of blocks:

```
{
    System.out.print("The answer is ");
    System.out.println(ans);
}
```

```
{  // This block exchanges the values of x and y
    int temp = x;   // declare temp and store x in it
    x = y;          // copy value of y into x
    y = temp;       // copy value of temp into y
}
```

IF statement

- An if statement has one of the forms:

```
if ( boolean-expression )
    statement
else
    statement
```

```
if ( boolean-expression )
    statement
```

IF statement and block

- As usual, each of the statement's in an if statement can be a block, so that an if statement often looks like:

```
if ( boolean-expression ) {
     statements
}
else {
     statements
}
```

```
if ( boolean-expression ) {
    statements
}
```

if-else ladder

- Example: not two, but three cases to check:

```
if (boolean-expression-1)
    statement-1
else
    if (boolean-expression-2)
       statement-2
    else
       statement-3
```

- Normally written as:

```
if (boolean-expression-1)
   statement-1
else if (boolean-expression-2)
   statement-2
else
   statement-3
```

if-else ladder

- Here is an example that will print out one of three different messages, depending on the value of a variable named temperature:

```
if (temperature < 50)
    System.out.println("It's cold.");
else if (temperature < 80)
    System.out.println("It's nice.");
else
    System.out.println("It's hot.");
```

if-else ladder

- You can go on stringing together "**else-if's**" to make multi-way branches with any number of cases:

```
if (boolean-expression-1)
    statement-1
else if (boolean-expression-2)
    statement-2
else if (boolean-expression-3)
    statement-3
      .
      . // (more cases)
      .
else if (boolean-expression-N)
    statement-N
else
    statement-(N+1)
```

if-else in first branch danger!!

- Check that:

```
if ( x > 0 )
   if (y > 0)
       System.out.println("First case");
else
   System.out.println("Second case");
```

- . . . is probably misunderstood

a switch as a special form of if-else ladder

- A **switch** statement has the form:

```
switch (expression) {
    case constant-1:
        statements-1
        break;
    case constant-2:
        statements-2
        break;
      .
      .   // (more cases)
      .
    case constant-N:
        statements-N
        break;
    default:  // optional default case
        statements-(N+1)
    } // end of switch statement
```

switch example

```
switch (N) {   // assume N is an integer variable
   case 1:
       System.out.println("The number is 1.");
       break;
    case 2:
    case 4:
    case 8:
       System.out.println("The number is 2, 4, or 8.");
       System.out.println("(That's a power of 2!)");
       break;
    case 3:
    case 6:
    case 9:
       System.out.println("The number is 3, 6, or 9.");
       System.out.println("(That's a multiple of 3!)");
       break;
    case 5:
       System.out.println("The number is 5.");
       break;
    default:
       System.out.println("The number is 7,");
       System.out.println(" or outside the range 1 to 9");
}
```

While LOOP

- A while loop has the form:

```
while (boolean-expression)
     statement
```

- Since the statement can be, and usually is, a block, many while loops have the form:

```
while (boolean-expression) {
     statements
}
```

While LOOP

- Here is an example of a while loop that simply prints out the numbers 1, 2, 3, 4, 5:

```
int number = 1;
while ( number < 6 ) {
      System.out.println(number);
      number = number + 1;
}
System.out.println("Done!");
```

Do…while loop

- A do…while loop has the form:

```
do
    statement
while (boolean-expression);
```

- Since the statement can be, and usually is, a block, many do…while loops have the form:

```
do  {
      statements
} while (boolean-expression);
```

Do…while loop example

- Pseudocode:

```
do {
     Play a Game
     Ask user if he wants to play another game
     Read the user's response
} while ( the user's response is yes );
```

- Real code:

```
boolean wantsToContinue;  // True if user wants to play
                          // again.
do {
    Checkers.playGame();
    TextIO.put("Do you want to play again? ");
    wantsToContinue = TextIO.getlnBoolean();
 } while (wantsToContinue == true);
```

Break and continue

- The syntax of the while and do..while loops allows you to test the continuation condition at either the beginning of a loop or at the end.
- Sometimes, it is more natural to have the test in the middle of the loop, or to have several tests at different places in the same loop.
- Java provides a general method for breaking out of the middle of any loop. It's called **the break statement**, which takes the form

```
break;
```

- When the computer executes a break statement in a loop, it will immediately jump out of the loop. It then continues on to whatever follows the loop in the program.

Example: break

```
while (true) {  // looks like it will run forever!
    TextIO.put("Enter a positive number: ");
    N = TextIO.getlnInt();
    if (N > 0)   // input is OK; jump out of loop
       break;
    TextIO.putln("Your answer must be > 0.");
}
// continue here after break
```

- A break statement terminates the loop that immediately encloses the break statement. It is possible to have nested loops, where one loop statement is contained inside another. If you use a break statement inside a nested loop, it will only break out of that loop, not out of the loop that contains the nested loop.

- There is a "labeled break" statement that allows you to specify which loop you want to break.

Continue

- The continue statement is related to break, but less commonly used. A continue statement tells the computer to skip the rest of the current iteration of the loop.

- However, instead of jumping out of the loop altogether, it jumps back to the beginning of the loop and continues with the next iteration (after evaluating the loop's continuation condition to see whether any further iterations are required).

For LOOP

- The for statement makes a common type of while loop easier to write.

- Many while loops have the general form:

```
initialization
while ( continuation-condition ) {
        statements
        update
}
```

For LOOP

- Example with while:

```
years = 0;  // initialize the variable years
 while ( years < 5 ) {   // condition for continuing loop
    interest = principal * rate;
    principal += interest;
    System.out.println(principal);
    years++;   // update the value of the variable, years
}
```

- This loop can be written as the following equivalent for statement:

```
for ( years = 0;  years < 5;  years++ ) {
    interest = principal * rate;
    principal += interest;
    System.out.println(principal);
 }
```

For LOOP

- The formal syntax of the for statement is as follows:

```
for ( initialization; continuation-condition; update )
    statement
```

- or, using a block statement:

```
for ( initialization; continuation-condition; update ) {
    statements
}
```

- The continuation-condition must be a boolean-valued expression. The initialization can be any expression, as can the update.
- Any of the three can be empty. If the continuation condition is empty, it is treated as if it were "true," so the loop will be repeated forever or until it ends for some other reason, such as a break statement.
- Some people like to begin an infinite loop with "for (;;)“ instead of "while (true)".

Nested loops

- How to print a multiplication table like follows?

```
 1   2   3   4   5   6   7   8   9  10  11  12
 2   4   6   8  10  12  14  16  18  20  22  24
 3   6   9  12  15  18  21  24  27  30  33  36
 4   8  12  16  20  24  28  32  36  40  44  48
 5  10  15  20  25  30  35  40  45  50  55  60
 6  12  18  24  30  36  42  48  54  60  66  72
 7  14  21  28  35  42  49  56  63  70  77  84
 8  16  24  32  40  48  56  64  72  80  88  96
 9  18  27  36  45  54  63  72  81  90  99 108
10  20  30  40  50  60  70  80  90 100 110 120
11  22  33  44  55  66  77  88  99 110 121 132
12  24  36  48  60  72  84  96 108 120 132 144
```

Nested loops

- Example code with one for loop inside another

```
for ( rowNumber = 1;  rowNumber <= 12;  rowNumber++ ) {
   for ( N = 1;  N <= 12;  N++ ) {
      // print in 4-character columns
      TextIO.put( N * rowNumber,  4 );
   }
   TextIO.putln();
}
```

Empty statement

- You can write extra semicolons if you like: these are treated as empty statements.

```
if  (x < 5)
  System.out.println("Hello"); ; ;
; x = x + 4 ; ;
```

is perfectly OK code

- Beware!

```
for (int i = 0; i < 10; i++);
   System.out.println("Hello");
```

does NOT print “Hello” 10 times!!

Statement types

- **declaration statement (for declaring variables)**
- **assignment statement**
- **subroutine call statement (including input/output routines)**
- other expression statement (such as "x++;")
- empty statement
- **block statement**
- **while statement**
- do..while statement
- **if statement**
- **for statement**
- switch statement
- break statement (found in loops and switch statements only)
- continue statement (found in loops only)
- **return statement (found in subroutine definitions only)**
- try..catch statement
- throw statement
- synchronized statement

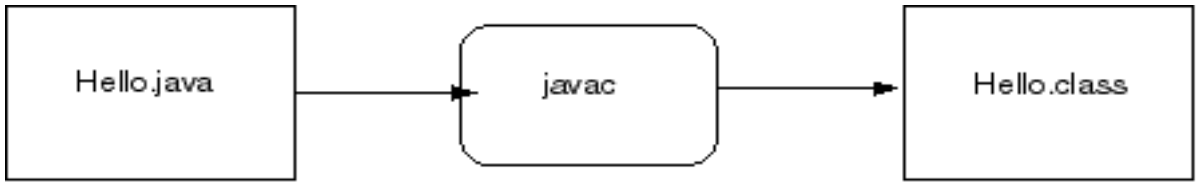
Programmi kirjutamise etapid:

- **getting the program text into the computer,**
- **compiling the program, and**
- **running the compiled program.**

Final step - running the program - either as

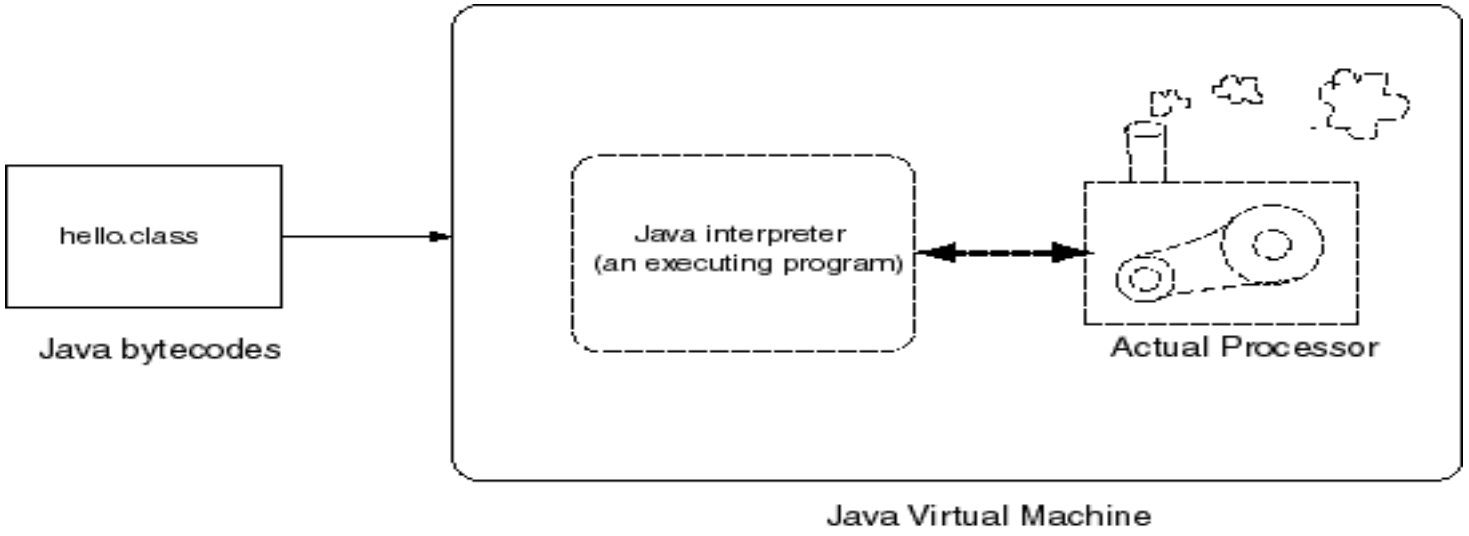
- Application - program running without a www browser
- Applet- program running in a www browser
- Servlet- program running in a (web) server

Compiler, bytecodes



Java Program Translation

Executing bytecodes



Java Bytecode Interpretation on a Virtual Machine

Java: APPLICATION

```
public class HelloWorld {

   // A program to display the message
   // "Hello World!" on standard output

   public static void main(String[] args) {
        System.out.println("Hello World!");
   }

}    // end of class HelloWorld
```

- kompileerime: **javac HelloWorld.java =>** HelloWorld.class
- paneme käsurealt käima: **java HelloWorld**

function called **main**, with a definition:

```
public static void main(String[] args) {
     statements
  }
```

Java: APPLET

```
import java.applet.*;
import java.awt.*  ;

public class Helloa extends Applet
{
   public void init() {
     resize(300, 500);
   }

   public void paint(Graphics g) {
     g.drawString("Hello", 10,50);
   }
}
```

kompileerime: javac Helloa.java => Helloa.class
HTML failis: Tere.html

```
See on Hello applet:

<applet code = "Helloa"  width=120 height=120>
</applet>

Hello applet loppes.
```

Java: SERVLET

```
package test;
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class HelloServlet extends HttpServlet {

   public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
         throws ServletException, IOException {

     PrintWriter out = res.getWriter();
     out.println("Hello, world!");
     out.close();
   }
}
```

kompileerime: javac HelloServlet.java => Helloa.class

Seejärel vajame J2EE serverirakendust (tomcat, jboss vms), mis selle servleti käima suudab panna.